Algorithmic Paradigms

**Greedy.** Build up a solution incrementally, myopically optimizing some local criterion.

**Dynamic programming.** Break up a problem into a series of overlapping sub-problems, and build up solutions to larger and larger sub-problems.

**Divide-and-conquer.** Break up a problem into sub-problems, solve each sub-problem independently, and combine solution to sub-problems to form solution to original problem.
Divide-and-Conquer

Divide-and-conquer.
- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.
- Break up problem of size $n$ into two equal parts of size $\frac{1}{2}n$.
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

Consequence.
- Brute force: $n^2$.
- Divide-and-conquer: $n \log n$.

Divide et impera.
Veni, vidi, vici.
- Julius Caesar
5.1 Mergesort
Mergesort

Mergesort.
- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.

Jon von Neumann (1945)

\[
\begin{array}{cccccccc}
A & L & G & O & R & I & T & H & M & S \\
A & L & G & O & R & I & T & H & M & S \\
A & G & L & O & R & H & I & M & S & T \\
A & G & H & I & L & M & O & R & S & T \\
\end{array}
\]

 divide \( O(1) \)
 sort \( 2T(n/2) \)
 merge \( O(n) \)
**Merging**

**Merge.**
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

\[\text{smallest}\]

\[
\begin{array}{cccc}
A & G & L & O & R \\
\end{array}
\]

\[
\begin{array}{cccc}
H & I & M & S & T \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{auxiliary array} \\
A & & & & \\
\end{array}
\]
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

auxiliary array
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

**Auxiliary array**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>G</td>
<td>L</td>
<td>O</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>I</td>
<td>M</td>
<td>S</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

|   |   |   |   |   |
|---|---|---|---|
| A | G | H | I |   |
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

辅助数组

A G L O R    H I M S T
A G H I L
Merging

**Merge.**

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.
Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

```
A G L O R
H I M S T
```

```
A G H I L M O R
```

auxiliary array
Merging

**Merge.**
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

```
A G L O R
H I M S T
```

```
A G H I L M O R S
```

**auxiliary array**
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

first half exhausted

smallest

A G L O R

H I M S T

A G H I L M O R S T

auxiliary array
Merging

**Merge.**
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

```
AGLOR
```
```
HIMST
```
```
AGHILMORSST
```
Merging

Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently?

- Linear number of comparisons.
- Use temporary array.

Challenge for the bored. In-place merge. [Kronrud, 1969]

using only a constant amount of extra storage
A Useful Recurrence Relation

**Def.** \( T(n) = \) number of comparisons to mergesort an input of size \( n \).

**Mergesort recurrence.**

\[
T(n) \leq \begin{cases} 
0 & \text{if } n = 1 \\
T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise}
\end{cases}
\]

**Solution.** \( T(n) = O(n \log_2 n) \).

**Assorted proofs.** We describe several ways to prove this recurrence. Initially we assume \( n \) is a power of \( 2 \) and replace \( \leq \) with \( = \).
Proof by Recursion Tree

\[ T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + n & \text{otherwise} 
\end{cases} \]

sorting both halves
merging
Proof by Telescoping

**Claim.** If $T(n)$ satisfies this recurrence, then $T(n) = n \log_2 n$.

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + n & \text{otherwise}
\end{cases}
\]

assumes $n$ is a power of 2

**Pf.** For $n > 1$:

\[
\frac{T(n)}{n} = \frac{2T(n/2)}{n} + 1
\]

\[
= \frac{T(n/2)}{n/2} + 1
\]

\[
= \frac{T(n/4)}{n/4} + 1 + 1
\]

\[\vdots\]

\[
= \frac{T(n/n)}{n/n} + 1 + \cdots + 1
\]

\[
= \log_2 n
\]
Proof by Induction

**Claim.** If $T(n)$ satisfies this recurrence, then $T(n) = n \log_2 n$.

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + n & \text{otherwise}
\end{cases}
\]

\[\text{assumes } n \text{ is a power of } 2\]

**Pf.** (by induction on $n$)

- **Base case:** $n = 1$.
- **Inductive hypothesis:** $T(n/2) = n/2 \log_2 (n/2)$.
- **Goal:** show that $T(n) = n \log_2 (n)$.

\[
T(n) = 2T(n/2) + n \\
= 2(n/2)\log_2(n/2) + n \\
= n\log_2 n - 1 + n \\
= n\log_2 n
\]
"Proof" by Guessing

Claim. If $T(n)$ satisfies this recurrence, then $T(n) = n \log_2 n$.

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + n & \text{otherwise}
\end{cases}
\]

NOT a Proof. (by guessing)

- Assume $T(n) = n \log_2 (n)$.
- Goal: plug in and check if this is true.

\[
\begin{align*}
n \log_2(n) &= T(n) = 2T(n/2) + n \\
&= 2(n/2) \log_2(n/2) + n \\
&= n(\log_2 n - 1) + n \\
&= n \log_2(n)
\end{align*}
\]

- Useful when figuring things out. To formalize (for homework etc), use induction.

assumes $n$ is a power of 2
Analysis of Mergesort Recurrence

**Claim.** If $T(n)$ satisfies the following recurrence, then $T(n) \leq n \lfloor \lg n \rfloor$.

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

**Pf.** (by induction on $n$)

- **Base case:** $n = 1$.
- **Define** $n_1 = \lceil n / 2 \rceil$, $n_2 = \lfloor n / 2 \rfloor$.
- **Induction step:** assume true for $1, 2, \ldots, n-1$.

$$T(n) \leq T(n_1) + T(n_2) + n \leq n_1 \lfloor \lg n_1 \rfloor + n_2 \lfloor \lg n_2 \rfloor + n \leq n_1 \lfloor \lg n_2 \rfloor + n_2 \lfloor \lg n_2 \rfloor + n = n \lfloor \lg n_2 \rfloor + n \leq n(\lfloor \lg n \rfloor - 1) + n = n \lfloor \lg n \rfloor$$

$$n_2 = \lceil n/2 \rceil \leq 2^\lceil \lg n \rceil / 2 = 2^\lfloor \lg n \rfloor / 2 \Rightarrow \lg n_2 \leq \lfloor \lg n \rfloor - 1$$
Chapter 5

Divide and Conquer (and Combine)
5.4 Closest Pair of Points
Closest Pair of Points

**Closest pair.** Given n points in the plane, find a pair with smallest Euclidean distance between them.

**Fundamental geometric primitive.**
- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control. Euclidean MST, Voronoi.
Closest Pair of Points

Closest pair. Given n points in the plane, find a pair with smallest Euclidean distance between them.

Brute force.
- Check all pairs of points p and q with $\Theta(n^2)$ comparisons.

1-D version.
- $O(n \log n)$ easy if points are on a line.

Assumption. No two points have same x coordinate.

↑

to make presentation cleaner
Closest Pair of Points: First Attempt

Divide.

Sub-divide region into 4 quadrants with \( n/4 \) points in each.
Closest Pair of Points: First Attempt

**Divide.** Sub-divide region into 4 quadrants.

**Obstacle.** Impossible to ensure \( n/4 \) points in each piece.
Closest Pair of Points

Algorithm.

- **Divide:** draw vertical line $L$ with $\frac{1}{2}n$ points on each side.
Closest Pair of Points

Algorithm.
- **Divide:** draw vertical line $L$ with $\frac{1}{2}n$ points on each side.
- **Conquer:** find closest pair in each side recursively.
Closest Pair of Points

**Algorithm.**
- **Divide:** draw vertical line L so that roughly \( \frac{1}{2} n \) points on each side.
- **Conquer:** find closest pair in each side recursively.
- **Combine:** find closest pair with one point in each side. \( \rightarrow \) seems like \( \Theta(n^2) \)
Closest Pair of Points

Note that the \textit{max relevant distance} < \(\delta\).
Closest Pair of Points

Note that the \textit{max relevant distance} \(< \delta\).

- Observation: only need to consider points within \(\delta\) of line \(L\).

\[
\delta = \min(12, 21)
\]
Closest Pair of Points

Note that the max relevant distance < $\delta$.

- Observation: only need to consider points within $\delta$ of line $L$.
- Sort points in $2\delta$-strip by their $y$ coordinate.
**Closest Pair of Points**

**Def.** Let $s_i$ be the point in the $2\delta$-strip, with the $i^{th}$ smallest $y$-coordinate.

**Claim.** If $|i - j| \geq 12$, then the distance between $s_i$ and $s_j$ is at least $\delta$.

**Pf.**
- No two points lie in same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta)$.

**Fact.** Still true if we replace 12 with 7.
Closest Pair of Points

Note that the max relevant distance $< \delta$.

- Observation: only need to consider points within $\delta$ of line $L$.
- Sort points in $2\delta$-strip by their $y$ coordinate.
- Only check distances within 11 positions in sorted list! $\quad \leftarrow \text{is } \Theta(n)$
Closest Pair Algorithm

Closest-Pair\( (p_1, \ldots, p_n) \) \{

\textbf{Compute} separation line \( L \) \( \quad O(n \log n) \)

\[ \delta_L = \text{Closest-Pair(left half)} \quad 2T(n / 2) \]
\[ \delta_R = \text{Closest-Pair(right half)} \]
\[ \delta = \min(\delta_L, \delta_R) \]

\textbf{Delete} points further than \( \delta \) from \( L \) \( \quad O(n) \)

\textbf{Sort} remaining points by \( y \)-coordinate. \( \quad O(n \log n) \)

\textbf{Scan} compare distance to next 11 neighbors. \( O(n) \)
\hspace{1cm} \text{If less than} \( \delta \), \text{update} \( \delta \).

\textbf{return} \( \delta \).
\}

\[ O(n \log n) \]
\[ 2T(n / 2) \]
\[ O(n) \]
\[ O(n \log n) \]
\[ O(n) \]
Closest Pair of Points: Analysis

Running time.

\[ T(n) \leq 2T(n/2) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n) \]

Q. Can we achieve \( O(n \log n) \)?

A. Yes. Don’t sort points from scratch each time.
   - Use two lists: X sorted by x-coordinate, and Y by y-coordinate.
   - Sort by merging two pre-sorted lists.
Closest Pair Algorithm

Closest-Pair($X_1, \ldots, X_n, Y_1, \ldots, Y_n$) {

    \textbf{Compute} separation line $L$ \hfill $O(n)$

    $\delta_L, X_L, Y_L = \text{Closest-Pair(left half)}$ \hfill $2T(n / 2)$
    $\delta_R, X_R, Y_R = \text{Closest-Pair(right half)}$
    $\delta = \min(\delta_1, \delta_2)$

    \textbf{Merge} $X_L$ and $X_R$ \hfill $O(n)$
    \textbf{Merge} $Y_L$ and $Y_R$

    \textbf{Delete} points further than $\delta$ from $L$ \hfill $O(n)$

    \textbf{Sort} remaining points by $y$-coordinate. \hfill $O(n \log n)$

    \textbf{Scan} compare distance to next 11 neighbors. $O(n)$
    \textbf{If} less than $\delta$, update $\delta$.

    \textbf{return} $\delta$.

}
Closest Pair of Points: Analysis

Running time.

Q. Can we achieve $O(n \log n)$?

A. Yes. Don't sort points from scratch each time.
   - Return two lists: $Y$ sorted by $y$-coordinate, and $X$ by $x$-coordinate.
   - Sort by merging two pre-sorted lists.

$$T(n) \leq 2T(n/2) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n)$$
5.3 Counting Inversions
Counting Inversions

Music site tries to match your song preferences with others.
- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.
- My rank: 1, 2, ..., n.
- Your rank: \( a_1, a_2, \ldots, a_n \).
- Songs \( i \) and \( j \) inverted if \( i < j \), but \( a_i > a_j \).

### Songs

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Me</strong></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td><strong>You</strong></td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Inversions
3-2, 4-2

Brute force: check all \( \Theta(n^2) \) pairs \( i \) and \( j \).
Applications

- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall's Tau distance).
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide:** separate list into two pieces.

![Divide-and-conquer diagram]

Divide: $O(1)$. 

1 5 4 8 10 2 6 9 12 11 3 7

1 5 4 8 10 2 6 9 12 11 3 7
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
- **Divide**: separate list into two pieces.
- **Conquer**: recursively count inversions in each half.

Divide: $O(1)$.
Conquer: $2T(n/2)$

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>10</th>
<th>2</th>
<th>6</th>
<th>9</th>
<th>12</th>
<th>11</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
</table>

5 blue-blue inversions

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

8 green-green inversions

5-4, 5-2, 4-2, 8-2, 10-2
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide:** separate list into two pieces.
- **Conquer:** recursively count inversions in each half.
- **Combine:** count inversions where $a_i$ and $a_j$ are in different halves, and return sum of three quantities.

\[
\begin{array}{cccccccccccc}
1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 & 12 & 11 & 3 & 7 \\
\end{array}
\]

Divide: $O(1)$.

\[
\begin{array}{cccccccccccc}
1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 & 12 & 11 & 3 & 7 \\
\end{array}
\]

Conquer: $2T(n / 2)$

- 5 blue-blue inversions
- 8 green-green inversions

Combine: ???

9 blue-green inversions
- 5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Total = $5 + 8 + 9 = 22$. 

Combine Step: Sorted Count

Sorted count.

- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.

```plaintext
i = 6

3 7 10 14 18 19

two sorted halves

2 11 16 17 23 25
```
Combine Step: Sorted Count

Sorted count.

- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.

\[
i = 6
\]

\[
3 \quad 7 \quad 10 \quad 14 \quad 18 \quad 19
\]

\[
6
\]

\[
2 \quad 11 \quad 16 \quad 17 \quad 23 \quad 25
\]

two sorted halves
Combine Step: Sorted Count

Sorted count.

- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.

\[
i = 6
\]

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array}
\]

two sorted halves

6
Combine Step: Sorted Count

Sorted count.

- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.

\[ i = 6 \]

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array}
\]

two sorted halves
Sorted count.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array}
\]

\[
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array}
\]

\( i = 5 \)
\( \downarrow \)
\( \downarrow \)
\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array}
\]

\( \frac{6}{2} \)

\( \frac{6}{2} \)

two sorted halves
Combine Step: Sorted Count

Sorted count.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.

```
i = 5

3 7 10 14 18 19

2 11 16 17 23 25

6
two sorted halves
```
Sorted count.

- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\downarrow & & & & & \\
2 & 11 & 16 & 17 & 23 & 25 \\
\downarrow & & & & & \\
6 & & & & &
\end{array}
\]

two sorted halves
Combine Step: Sorted Count

Sorted count.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.

```
i = 4
  ↓
3 7 10 14 18 19  ↓
2 11 16 17 23 25
```
two sorted halves

6
**Combine Step: Sorted Count**

**Sorted count.**
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.

<table>
<thead>
<tr>
<th>3</th>
<th>7</th>
<th>10</th>
<th>14</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
</table>

\[ i = 3 \]

\[ \downarrow \]

<table>
<thead>
<tr>
<th>2</th>
<th>11</th>
<th>16</th>
<th>17</th>
<th>23</th>
<th>25</th>
</tr>
</thead>
</table>

6

**two sorted halves**
Combine Step: Sorted Count

Sorted count.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.

\[ i = 3 \]

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array}
\]

\[
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & & & &
\end{array}
\]

two sorted halves
Combine Step: Sorted Count

Sorted count.

- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\downarrow & & & & & \\
6 & & & & & 3
\end{array} \quad \begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\downarrow & & & & & \\
6 & & & & & 3
\end{array} \quad \text{two sorted halves}
\]
**Combine Step: Sorted Count**

**Sorted count.**

- Given two sorted halves, count number of inversions where \(a_i\) and \(a_j\) are in different halves.

![Diagram](image-url)

- \(i = 3\)
- Two sorted halves:
  - Left half: 3, 7, 10, 14, 18, 19
  - Right half: 2, 11, 16, 17, 23, 25

\(i = 3\)
Combine Step: Sorted Count

Sorted count.

- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.

\[
i = 2
\]

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 \\
\end{array}
\] two sorted halves
Combine Step: Sorted Count

Sorted count.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\downarrow & & & & & \\
3 & 7 & 10 & 14 & 18 & 19
\end{array}
\quad
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 & & & \\
2 & 11 & 16 & 17 & 23 & 25
\end{array}
\quad
\text{two sorted halves}
\]
Sorted count.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$
  are in different halves.

\[
\begin{array}{c}
i = 2 \\
\downarrow \\
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
6 & 3 & 2
\end{array}
\end{array}
\quad \begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
two sorted halves
\end{array}
\]
Combine Step: Sorted Count

Sorted count.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.

\[ i = 2 \]

<table>
<thead>
<tr>
<th>3</th>
<th>7</th>
<th>10</th>
<th>14</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 2 | 11 | 16 | 17 | 23 | 25 |

Two sorted halves
Combine Step: Sorted Count

Sorted count.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.

\[
\begin{array}{ccccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\downarrow & & & & & \\
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 & 2 & & \\
\end{array}
\]

two sorted halves
Combine Step: Sorted Count

Sorted count.

- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.

\[
\begin{array}{cccccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\downarrow &   &   &   &   &   \\
6 & 3 & 2 & 2 & & & \\
\end{array}
\quad \begin{array}{cccccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\downarrow &   &   &   &   &   \\
two sorted halves & & & & & & &
\end{array}
\]
Combine Step: Sorted Count

Sorted count.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.

\[ \begin{align*}
\text{two sorted halves} \\
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
6 & 3 & 2 & 2
\end{array}
\end{align*} \]
**Combine Step: Sorted Count**

Sorted count.

- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.

\[
\begin{array}{ccccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array}
\quad \downarrow \\
\begin{array}{ccccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 & 2 \\
\end{array}
\quad \text{two sorted halves}
\]
Combine Step: Sorted Count

Sorted count.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.

\[
\begin{array}{cccccccc}
\text{first half} & \downarrow & i = 0 & \downarrow & \text{two sorted halves} \\
3 & 7 & 10 & 14 & 18 & 19 & 2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 & 2
\end{array}
\]
Combine Step: Sorted Count

Sorted count.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.

\[
i = 0
\]

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 & 2 & 0 &
\end{array}
\]

two sorted halves
Combine Step: Sorted Count

Sorted count.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.

\[
\begin{array}{c}
\text{i = 0} \\
\downarrow \\
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
| & | & | & | & | & \\
6 & 3 & 2 & 2 & 0 & \\
\end{array}
\end{array}
\]

two sorted halves
Combine Step: Sorted Count

**Sorted count.**

- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.

\[
\begin{array}{ccccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\hline
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 & 2 & 0 & 0 \\
\end{array}
\]

Two sorted halves

i = 0
Combine Step: Sorted Count

Sorted count.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.

\[
\begin{array}{c}
\text{i = 0} \\
\downarrow \\
\begin{array}{cccccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 & 2 & 0 & 0 \\
\end{array}
\end{array}
\]

two sorted halves

Total: \( 6 + 3 + 2 + 2 + 0 + 0 = 13 \)
Counting Inversions: Combine

**Combine:** count blue-green inversions

- Assume each half is sorted.
- Count inversions where \( a_i \) and \( a_j \) are in different halves.
- Merge two sorted halves into sorted whole.

\[ \rightarrow \text{to maintain sorted invariant} \]
**Combine Step: Merge and Count**

**Merge and count step.**
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[
i = 6
\]

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\downarrow & & & & & \\
\end{array}
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\downarrow & & & & & \\
\end{array}
\]

Two sorted halves

Auxiliary array

Total:
Combine Step: Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[ i = 6 \]

\[
\begin{array}{cccccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\downarrow & & & & & & & \\
\end{array}
\begin{array}{cccccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
6 & & & & & & & \\
\end{array}
\]

two sorted halves

\[
\begin{array}{cccccccc}
2 & & & & & & & \\
\end{array}
\]

auxiliary array

Total: 6
Combine Step: Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

![Diagram]

- $i = 6$

  \[
  \begin{array}{cccccc}
  3 & 7 & 10 & 14 & 18 & 19 \\
  \end{array}
  \quad \downarrow \quad \downarrow \quad \begin{array}{cccccc}
  2 & 11 & 16 & 17 & 23 & 25 \\
  \end{array}
  
  \text{two sorted halves}

  \begin{array}{cccccc}
  2 & \ & \ & \ & \ & \text{auxiliary array} \\
  \end{array}

  \text{Total: 6}
Combine Step: Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[
i = 6
\]
\[
\begin{array}{cccccccc}
3 & 7 & 10 & 14 & 18 & 19 & \downarrow & \downarrow \\
2 & 11 & 16 & 17 & 23 & 25 & \text{two sorted halves} & \text{auxiliary array}
\end{array}
\]

Total: 6
Combine Step: Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array}
\]

two sorted halves

\[
\begin{array}{cccccc}
2 & 3 \\
\end{array}
\]

auxiliary array

Total: 6
**Combine Step: Merge and Count**

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[
\begin{array}{ccccccc}
3 & 7 & 10 & 14 & 18 & 19 & \text{two sorted halves} \\
\downarrow & & & & & & 6 \\
2 & 11 & 16 & 17 & 23 & 25 & \text{auxiliary array} \\
\end{array}
\]

Total: 6
Combine Step: Merge and Count

**Merge and count step.**
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

<table>
<thead>
<tr>
<th>i = 4</th>
<th>3 7 10 14 18 19</th>
<th>2 11 16 17 23 25</th>
<th>two sorted halves</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 3 7</td>
<td></td>
<td>auxiliary array</td>
</tr>
</tbody>
</table>

**Total:** 6
Combine Step: Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[ i = 4 \]

\[ \begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array} \]

\[ \begin{array}{cccccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array} \]

two sorted halves

\[ \begin{array}{cccccc}
2 & 3 & 7 & 10 & & \\
\end{array} \]

auxiliary array

Total: 6
Combine Step: Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[
i = 3
\]

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\downarrow & & & \downarrow & & \\
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array}
\]

two sorted halves

\[
\begin{array}{cccccc}
2 & 3 & 7 & 10 & & & \\
\end{array}
\]

auxiliary array

Total: 6
Combine Step: Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array}
\quad \quad \quad \quad \quad \quad \quad
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array}
\]

\[
i = 3
\quad \quad \quad \quad \quad \quad \quad
\]

Total: $6 + 3$
**Combine Step: Merge and Count**

**Merge and count step.**

- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
\end{array}
\]

\[
\begin{array}{cccccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 \\
\end{array}
\]

\[
\begin{array}{cccccc}
2 & 3 & 7 & 10 & 11 \\
\end{array}
\]

\[
\text{Total: } 6 + 3
\]


Combine Step: Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[
\begin{array}{cccccc}
3 & 7 & 10 & \textbf{14} & 18 & 19 \\
\end{array}
\quad \downarrow \quad \downarrow
\begin{array}{cccccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3
\end{array}
\]

Two sorted halves

\[
\begin{array}{cccccc}
2 & 3 & 7 & 10 & 11 & 14
\end{array}
\]

Auxiliary array

Total: 6 + 3
Combine Step: Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[
i = 2 \\
\downarrow \\
3 \ 7 \ 10 \ 14 \ 18 \ 19 \\
\downarrow \\
2 \ 11 \ 16 \ 17 \ 23 \ 25
\]

Total: 6 + 3
Combine Step: Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[
\begin{array}{cccccccc}
3 & 7 & 10 & 14 & 18 & 19 & 2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 \\
\end{array}
\]

Total: 6 + 3 + 2
Combine Step: Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \(a_i\) and \(a_j\) are in different halves.
- Combine two sorted halves into sorted whole.

```
3  7  10  14  18  19
\(i = 2\)
```

```
2  11  16  17  23  25
6  3  2
two sorted halves
```

```
2  3  7  10  11  14  16
```

```
Total: 6 + 3 + 2
```

auxiliary array
Combine Step: Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

![Diagram showing merging and counting process](image)

Total: $6 + 3 + 2 + 2$
**Combine Step: Merge and Count**

Merge and count step.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[
\begin{align*}
\text{i} &= 2 \\
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
& & & & & \\
\end{array}
\quad \begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 & 2 \\
\end{array}
\quad \text{two sorted halves}
\end{align*}
\]

\[
\begin{array}{cccccc}
2 & 3 & 7 & 10 & 11 & 14 & 16 & 17 \\
\end{array}
\quad \text{auxiliary array}
\]

Total: \(6 + 3 + 2 + 2\)
**Combine Step: Merge and Count**

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>7</th>
<th>10</th>
<th>14</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>i = 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>11</td>
<td>16</td>
<td>17</td>
<td>23</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Total:** $6 + 3 + 2 + 2$
Combine Step: Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\downarrow & & & & & \\
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 & 2 & & \\
\end{array}
\]

\[
\begin{array}{cccccccc}
2 & 3 & 7 & 10 & 11 & 14 & 16 & 17 & 18 \\
\end{array}
\]

Total: \( 6 + 3 + 2 + 2 \)
Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

```
3 7 10 14 18 19
2 11 16 17 23 25
```

```
2 3 7 10 11 14 16 17 18 19
```

Total: $6 + 3 + 2 + 2$
Combine Step: Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\downarrow & & & & & \\
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array}
\]

\[
\begin{array}{cccccc}
2 & 3 & 7 & 10 & 11 & 14 & 16 & 17 & 18 & 19 \\
6 & 3 & 2 & 2 \\
\end{array}
\]

Total: $6 + 3 + 2 + 2$
Combine Step: Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[ \text{Total: } 6 + 3 + 2 + 2 + 0 \]
Combine Step: Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[
\begin{align*}
&\quad i = 0 \\
&\downarrow \\
3 & 7 & 10 & 14 & 18 & 19 & \quad 2 & 11 & 16 & 17 & 23 & 25 \\
&\quad & \quad & 6 & 3 & 2 & 2 & 0 \\
&\quad 2 & 3 & 7 & 10 & 11 & 14 & 16 & 17 & 18 & 19 & 23 \\
&\downarrow \\
&\quad \text{two sorted halves} \\
&\quad \text{auxiliary array} \\
\end{align*}
\]

Total: \( 6 + 3 + 2 + 2 + 0 \)
Combine Step: Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[
\begin{align*}
\text{i = 0} & \\
\downarrow & \\
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
6 & 3 & 2 & 2 & 0 & 0 \\
\end{array} & \\
\downarrow & \\
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
2 & 0 & 0 & & & \\
\end{array} & \\
\text{two sorted halves} & \\
\downarrow & \\
\begin{array}{cccccc}
2 & 3 & 7 & 10 & 11 & 14 & 16 & 17 & 18 & 19 & 23 & 25 \\
& & & & & & & & & & & & \\
\end{array} & \\
\text{auxiliary array} & \\
\downarrow & \\
\text{Total: 6 + 3 + 2 + 2 + 0 + 0} &
\]
Combine Step: Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[
\begin{array}{cccccccc}
3 & 7 & 10 & 14 & 18 & 19 & \\
\downarrow & & & & & & \\
2 & 11 & 16 & 17 & 23 & 25 & \\
& 6 & 3 & 2 & 2 & 0 & 0 &
\end{array}
\]
\[
\begin{array}{cccccccc}
2 & 3 & 7 & 10 & 11 & 14 & 16 & 17 & 18 & 19 & 23 & 25 &
\end{array}
\]

Total: \( 6 + 3 + 2 + 2 + 0 + 0 = 13 \)
Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.
Post-condition. [Sort-and-Count] L is sorted.

Sort-and-Count(L) {
    if list L has one element
        return 0 and the list L

    Divide the list into two halves A and B
    \((r_A, A) \leftarrow \text{Sort-and-Count}(A)\)
    \((r_B, B) \leftarrow \text{Sort-and-Count}(B)\)
    \((r, L) \leftarrow \text{Merge-and-Count}(A, B)\)

    return \(r = r_A + r_B + r\) and the sorted list L
}
Counting Inversions: Combine

Combine: count blue-green inversions
  - Assume each half is sorted.
  - Count inversions where $a_i$ and $a_j$ are in different halves.
  - Merge two sorted halves into sorted whole.

$$T(n) \leq T\left(\left\lfloor n/2 \right\rfloor \right) + T\left(\left\lceil n/2 \right\rceil \right) + O(n) \Rightarrow T(n) = O(n \log n)$$
Chapter 5
Divide and Conquer
5.5 Polynomial (Integer) Multiplication
Polynomial Multiplication

Multiplication of two n-degree polynomials is $O(n^2)$, addition is $O(n)$.

Consider the multiplication of two linear equations:
- Takes 4 multiplications and 1 addition
  \[(a + bx)(c + dx) = ac + (bc+ad)x + bdx^2\]

Consider two polynomial equations of degree $n ( = 2^k)$
- Divide and Conquer Approach: Use degree 1 case above
  
  \[p_1 = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n\]
  \[= (a_0 + \ldots + a_{n/2-1}x^{n/2-1}) + (a_{n/2} + \ldots + a_nx^{n/2})x^{n/2}\]
  \[= A + Bx^{n/2}\]

  \[p_2 = c_0 + c_1x + c_2x^2 + \ldots + c_nx^n\]
  \[= (c_0 + \ldots + c_{n/2-1}x^{n/2-1}) + (c_{n/2} + \ldots + c_nx^{n/2})x^{n/2}\]
  \[= C + Dx^{n/2}\]
Consider two polynomial equations of degree $n$

- **Divide and Conquer Approach:**
  
  
  $p_1 = A + Bx^{n/2}$
  
  $p_2 = C + Dx^{n/2}$
  
  $p_1p_2 = AC + (BC + AD)x^{n/2} + BDx^n$

- $A$, $B$, $C$, and $D$ are polynomials of degree $\leq n/2$.

- Recursively compute $AC$, $BC$, $AD$, and $BD$.

$$T(n) \leq 4T\left(\frac{n}{2}\right) + O(n) \quad \Rightarrow \quad T(n) = O(n^2)$$
Can we do fewer than 4 multiplications?

Try:

\[(a + bx)(c + dx) = ac + (bc+ad)x + bdx^2\]

\[= r + sx + tx^2\]

where

\[s = bc + ad\]

\[= (ac - ac) + bc + ad + (bd - bd)\]

\[= (a + b)(c + d) - r - t\]

\[= ef - r - t\]

Computing \(s\), \(r\), and \(t\) requires 3 multiplications and 4 additions.
Polynomial Multiplication

Consider two polynomial equations of degree $n$

- **Divide and Conquer Approach**: Use improved degree 1 case
  
  $p_1p_2 = R + Sx^{n/2} + Tx^n$

- **Recursively compute**:
  - $R = AC$
  - $T = BC$
  - $S = (A+B)(C+D) - R - T$

\[
T(n) = 3T(n/2) + O(n) \\
\Rightarrow T(n) = O(n^{\log_2 3}) \approx O(n^{1.58})
\]
In-Class Exercise

Solve the following recursions.
Assume $T(1) = c$. May make simplifying assumption for $n$ (i.e. $= 2^k$).

- $T(n) = T(n-1) + n$
- $T(n) = 5T(n/5) + 1$
- $T(n) = 2T(n-2) + 2$
- $T(n) = 3T(2n/3) + n$
Chapter 5

Divide and Conquer (and Combine)
Stooge Sort
Improved runtime for sort?

- Mergesort:

\[ T(n) \leq 2T\left(\frac{n}{2}\right) + O(n) \quad \Rightarrow \quad T(n) = O(n \log n) \]

- perhaps if we do less than \(O(n)\) “combine” work
Improved runtime for sort?

Stooge-Sort (L) {
    if list L has one element
        return L

    n = length(L)

    L[1,...,2n/3] ← Stooge-Sort(L[1,...,2n/3])
    L[n/3,...,n] ← Stooge-Sort(L[n/3,...,n])
    L[1,...,2n/3] ← Stooge-Sort(L[1,...,2n/3])

    return L
}

\[ T(n) = 3T(2n/3) + O(1) \]
Stooge Sort

**Improved runtime for sort?**

- Prove correctness by strong induction.

- Solve the recurrence:

\[ T(n) = 3T\left(\frac{2n}{3}\right) + O(1) \]
Master Theorem

**NOTE:** We use a more general theorem in class than in the uploaded chapter. Please use the version on the next slide for the homework and final.
The Master Theorem applies to recurrences of the form

\[ T(n) = aT\left(\frac{n}{b}\right) + f(n) \]

where \( a \geq 1 \) and \( b > 1 \) are constants, \( T(1) = c \) is a constant, and \( f(n) > 0 \).

1. If \( f(n) \in \mathcal{O}(n^{\log_b(a)} - \epsilon) \) with a constant \( \epsilon > 0 \),
   then \( T(n) \in \Theta(n^{\log_b(a)}) \).

2. If \( f(n) \in \Theta(n^{\log_b(a)} \log^k n) \) with a constant \( k \geq 0 \),
   then \( T(n) \in \Theta(n^{\log_b(a)} \log^{k+1} n) \).

3. If \( f(n) \in \Omega(n^{\log_b(a)} + \epsilon) \) with a constant \( \epsilon > 0 \), and \( f(n) \) satisfies *,
   then \( T(n) \in \Theta(f(n)) \).
3. If \( f(n) \in \Omega(n^{\log_b(a)} + \varepsilon) \) with \( \varepsilon > 0 \), and \( f(n) \) satisfies *,
then \( T(n) \in \Theta(f(n)) \).

* The regularity condition: \( af(n/b) \leq cf(n) \) for some constant \( c < 1 \) and all sufficiently large \( n \).

Works: polynomials, logs, exponentials, combinations thereof.
Does not work: \( n(2-\cos(n)) \)
The Master Theorem applies to recurrences of the form
\[ T(n) = aT\left(\frac{n}{b}\right) + f(n) \]

\[ n = b^k \]
\[ k = \log_b n \]

### Diagram:

- **Size:** \( b^k \), \( a^0 \), \( a^0 f(b^k) \)
- **Number:** \( a^1 \), \( a^1 f(b^{k-1}) \)
- **Work:** \( a^2 \), \( a^2 f(b^{k-2}) \)
- \( \vdots \)
- \( b^0 \), \( a^k \), \( a^k f(b^0) \)

Total \( = \sum a^i f(b^{k-i}) \)
The Master Theorem Proof

Proof of Case 1: \( f(n) \in O(n^{\log_b(a) - \epsilon}) \) with \( \epsilon > 0 \),
then \( T(n) \in \Theta(n^{\log_b(a)}) \).

Let \( k = \log_b n \)

Total = \( \sum_{i=0}^{k} a^i f(b^{(k-i)}) \)

= \( \sum_{i=0}^{k} a^i (b^{(k-i)})^{(\log_b(a)-\epsilon)} \)

= \( \sum_{i=0}^{k} a^i (b^{(\log_b(a)-\epsilon)})^{(k-i)} \)

= \( \sum_{i=0}^{k} a^i (b^{\log_b(a)})^{(k-i)} (b^{-\epsilon})^{(k-i)} \)

= \( \sum_{i=0}^{k} a^i a^{(k-i)} (1/b^\epsilon)^{(k-i)} \)

= \( a^k \sum_{i=0}^{k} (1/b^\epsilon)^{(k-i)} \)

\leq a^k = (b^{\log_b a})^k

= (b^k)^{\log_b a} = n^{\log_b a}
The Master Theorem Proof

**Proof of Case 3.** If \( f(n) \in \Omega(n^{\log_b(a)} + \epsilon) \) with \( \epsilon > 0 \), and \( f(n) \) satisfies \(*\), then \( T(n) \in \Theta(f(n)) \).

Let \( k = \log_b n \)

**Total** = \( \sum_{i=0}^{k} a^i f(b^{(k-i)}) \)

\[
= \sum_{i=0}^{k} a^i (b^{(k-i)})^{(\log_b(a) + \epsilon)}
\]

\[
= \sum_{i=0}^{k} a^i (b^{(\log_b(a) + \epsilon)})^{(k-i)}
\]

\[
= \sum_{i=0}^{k} a^i (b^{\log_b(a)})^{(k-i)} (b^\epsilon)^{(k-i)}
\]

\[
= \sum_{i=0}^{k} a^i a^{(k-i)} (b^\epsilon)^{(k-i)}
\]

\[
= a^k \sum_{i=0}^{k} (b^\epsilon)^{(k-i)}
\]

\[
= a^k (b^\epsilon)^{(k+1)}
\]

\[
= n^{\log_b a} (b^{(k+1)})^\epsilon
\]

\[
= n^{\log_b a + \epsilon}
\]
The Master Theorem Proof

**Proof of Case 2.** If \( f(n) \in \Theta(n^{\log_b(a)} \log^m n) \) with \( m \geq 0 \),
then \( T(n) \in \Theta(n^{\log_b(a)} \log^{k+1} n) \).

Let \( k = \log_b n \)

**Total** = \( \sum_{i=0}^{k} a^i f(b^{k-i}) \)

\[
= \sum_{i=0}^{k} a^i (b^{k-i})^{\log_b(a)} \log^m (b^{k-i})
\]

\( \leq \sum_{i=0}^{k} a^i (b^{\log_b(a)})^{(k-i)} \log^m (b^k) \)

\( = \log^m(n) \sum_{i=0}^{k} a^i a^{(k-i)} \)

\( = a^k \log^m(n) \sum_{i=0}^{k} 1 \)

\( = a^k k \log^m(n) \)

\( \leq n^{\log_b a} \log_b(n) \log^m(n) \)

\( = n^{\log_b a} \log^{m+1} n \)
The Master Theorem

The **Master Theorem** applies to recurrences of the form

\[ T(n) = aT\left(\frac{n}{b}\right) + f(n) \]

where \( a \geq 1 \) and \( b > 1 \) are constants, \( T(1) = c \) is a constant, and \( f(n) > 0 \).

**Note:** We proved this only for integers \( a \) and \( b \), and \( n = b^k \). Works for all real-valued \( a \geq 1 \), \( b > 1 \), and all \( n \) (i.e. floors and ceilings ok).

*May use without proof in homeworks etc! (but make sure conditions hold...)*
Examples

The Master Theorem: \( T(n) = aT(n/b) + f(n) \), where \( a \geq 1 \) and \( b > 1 \) are constants, \( T(1) = c \) is a constant, and \( f(n) > 0 \).

1. If \( f(n) \in O(n^{\log_b(a) - \varepsilon}) \), then \( T(n) \in \Theta(n^{\log_b(a)}) \).

2. If \( f(n) \in \Theta(n^{\log_b(a)} \log^k n) \) for \( k \geq 0 \), then \( T(n) \in \Theta(n^{\log_b(a)} \log^{k+1} n) \).

3. If \( f(n) \in \Omega(n^{\log_b(a) + \varepsilon}) \) and \( f(n) \) satisfies *, then \( T(n) \in \Theta(f(n)) \).

Solve the following recursions when \( T(1) = c \):

- \( T(n) = 4T(n/2) + n \)
- \( T(n) = 4T(n/2) + n^2 \)
- \( T(n) = 4T(n/2) + n^2 \log n \)
- \( T(n) = 4T(n/2) + n^3 \)
Stooge Sort

Improved runtime for sort?
  - Solve the recurrence:

\[
T(n) = 3T\left(\frac{2n}{3}\right) + O(1)
\]

\[\Rightarrow T(n) = O(n^{\log_{3/2} 3}) \approx O(n^{2.71})\]
The Master Theorem: $T(n) = aT(n/b) + f(n)$

1. If $f(n) \in O(n^{\log_b(a) - \epsilon})$, then $T(n) \in \Theta(n^{\log_b(a)})$.
2. If $f(n) \in \Theta(n^{\log_b(a)} \log^k n)$, then $T(n) \in \Theta(n^{\log_b(a)} \log^{k+1} n)$.
3. If $f(n) \in \Omega(n^{\log_b(a) + \epsilon})$ and $f(n)$ satisfies $\ast$, then $T(n) \in \Theta(f(n))$.

Solve the following recursions when $T(1) = 42$:

- $T(n) = 3T(n/3) + \sqrt{n}$
- $T(n) = T(n/2) + 2^n$
- $T(n) = 2.5T(n/2.5) + n \log n$
- $T(n) = 6T(n/3) + n^2 \log n$
- $T(n) = 3T(n/3) + 5n$